รายงานผลโครงงานการออกแบบวงจรเครื่องคิดเลขโดยใช้ภาษา VHDL

ที่สามารถใช้งาน การบวก การลบ การคูณ และการหาร ตัวเลขฐานสองขนาด N บิต

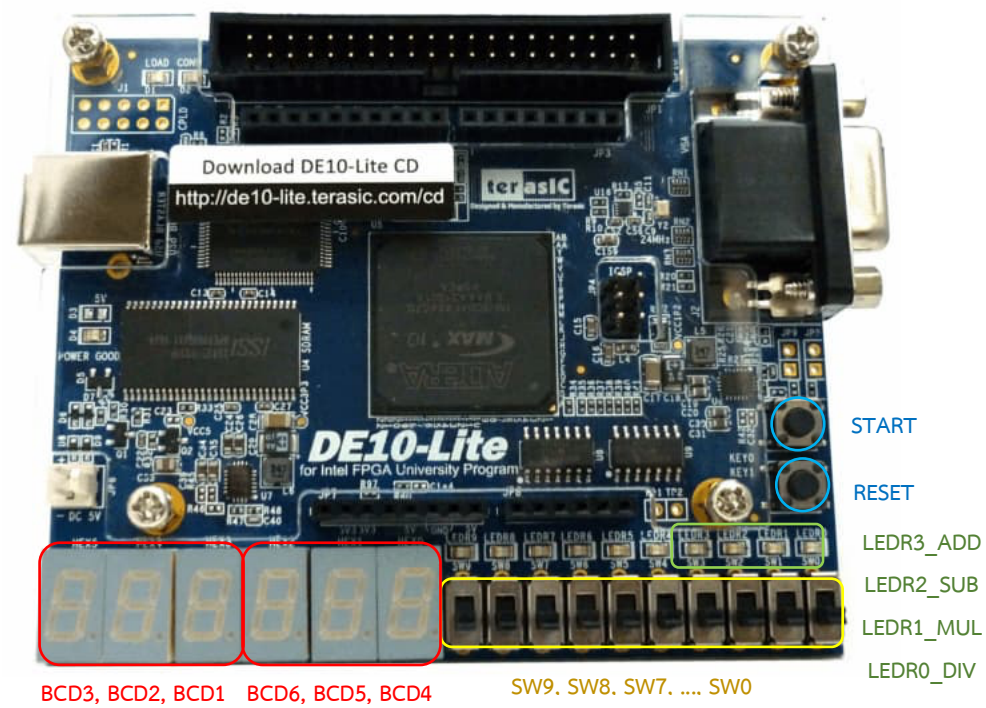
สมาชิก: 1. คุณาพจน์ สุทธินนท์ (65-010126-3003-4)
2. ภูตะวัน จันทร์เรือง (65-010126-3014-0)

ในรายงานฉบับนี้ ประกอบไปด้วย

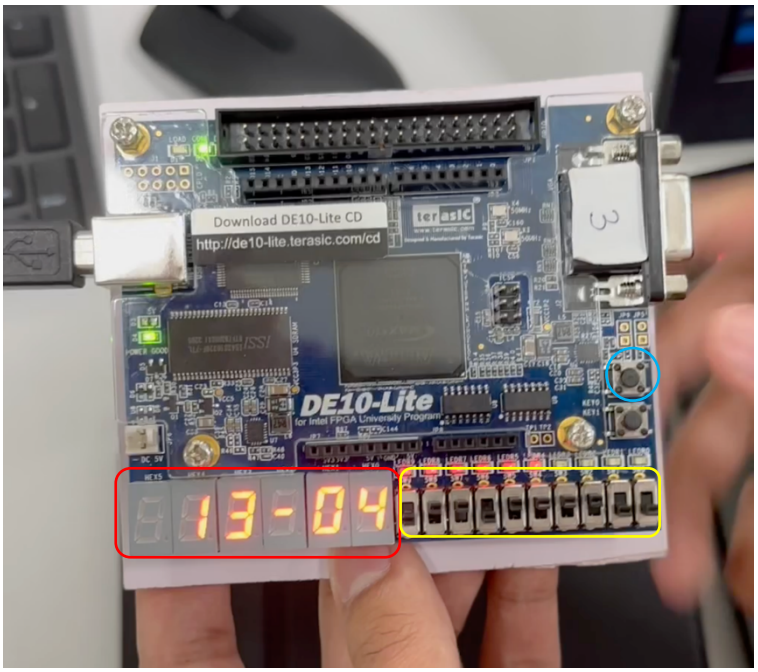
- สาธิตวิธีการใช้งานเบื้องต้น
- ตัวอย่างผลลัพธ์ที่ได้จากการใช้งาน
- การออกแบบวงจร

สาธิตวิธีการใช้งานเบื้องต้น

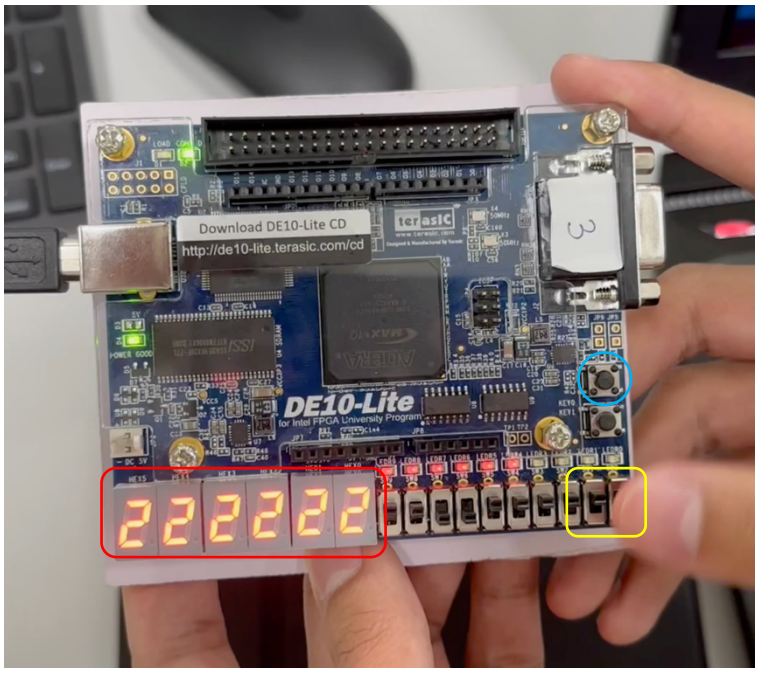
กำหนดให้ส่วนประกอบต่าง ๆ แทนชื่อดังนี้



1. เมื่อเริ่มใช้งาน เมนูแรกคือการเลือกตัวเลข โดย BCD2, BCD1, BCD5 และ BCD4 จะแสดงเลขฐานสิบสองหลัก 2 จำนวนตามลำดับ อ้างอิงจากการใช้ SW9-0 เป็นสวิตช์สำหรับกรอกเลข โดยเป็นเลขฐานสองชนิด Signed หากจำนวนที่กรอกมี Sign bit (SW9 และ SW4) เป็นเลข 1 จะส่งผลให้ BCD3 และ/หรือ BCD6 จะมีเครื่องหมายลบกำกับ

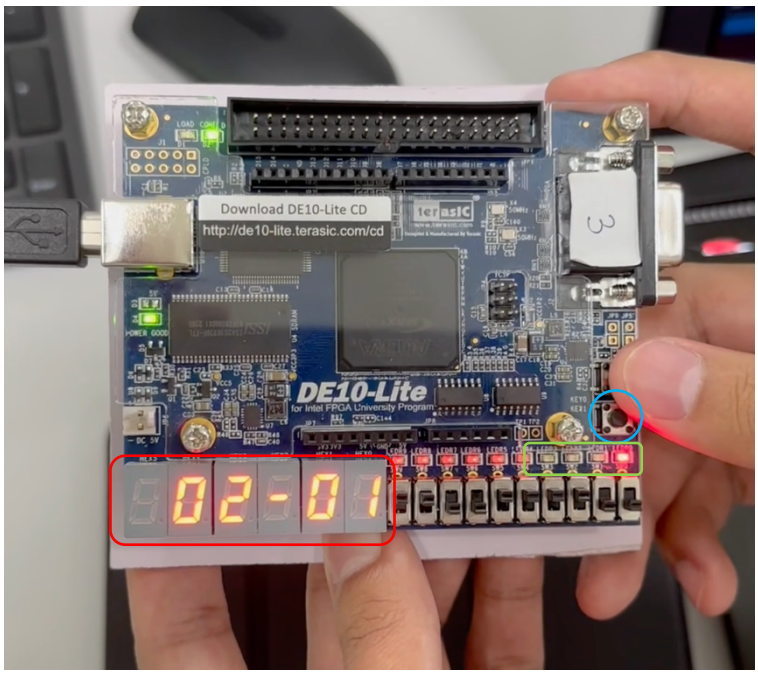


2. เมื่อกดปุ่ม START แล้วปล่อย จะเข้าสู่เมนูการเลือก Operator โดย BCD ทุกหลักจะแสดงเลขเป็นเลข 0, 1, 2 หรือ 3 สอดคล้องกับเลขฐานสอนที่ป้อนจาก SW1 และ SW0 กำหนดให้แต่ละจำนวนสื่อความหมายดังนี้
    a. ‘000000’ เป็นการหาร A ÷ B
    b. ‘111111’ เป็นการคูณ A × B
    c. ‘222222’ เป็นการลบ A - B
    d. ‘333333’ เป็นการบวก A + B

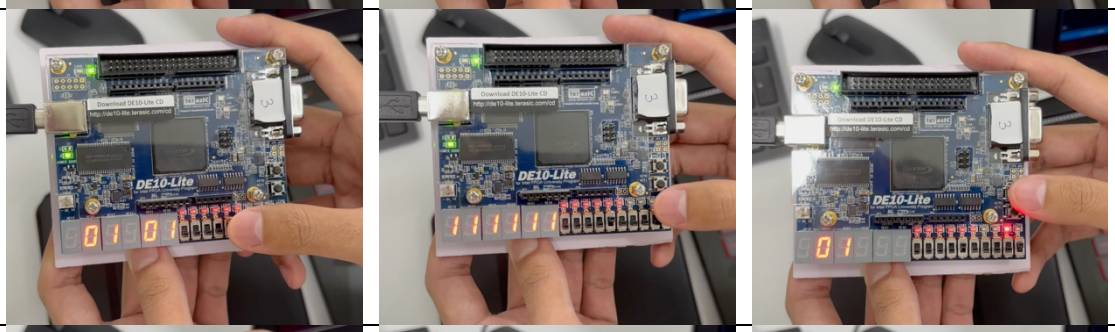
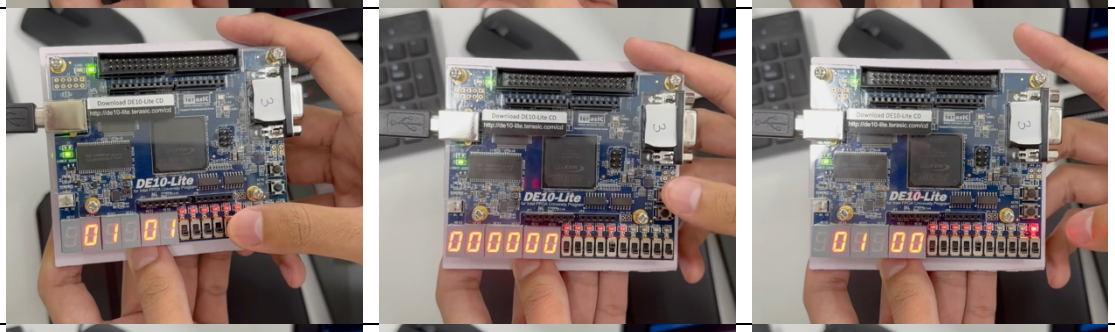


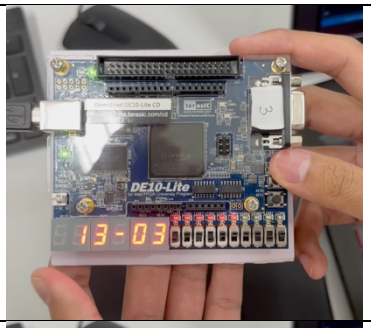
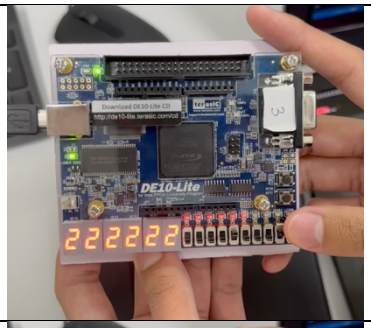
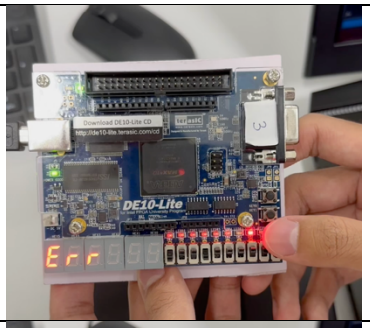
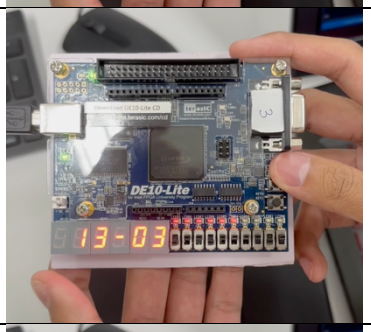
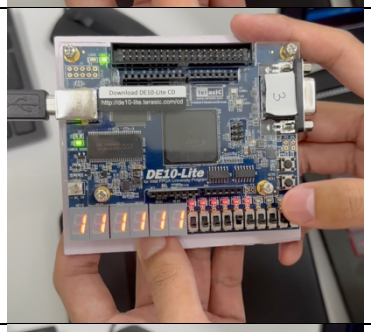
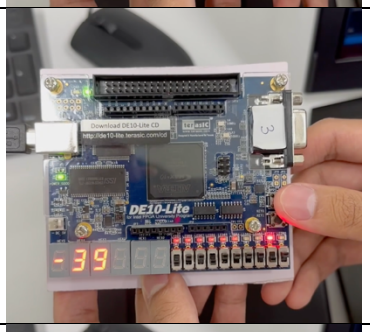
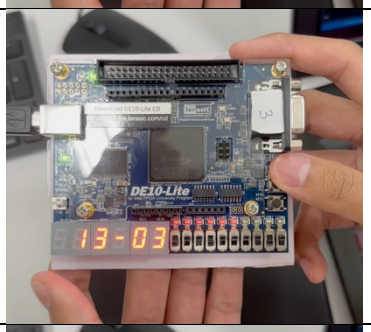
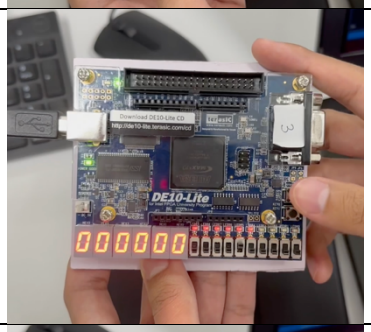
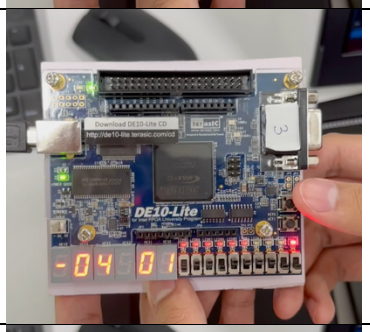
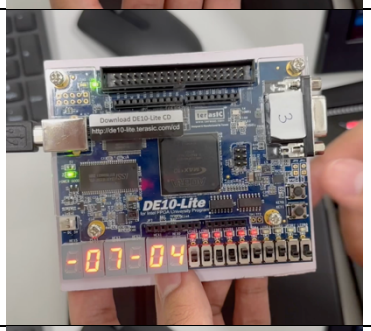
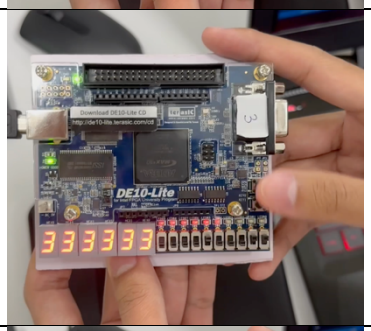
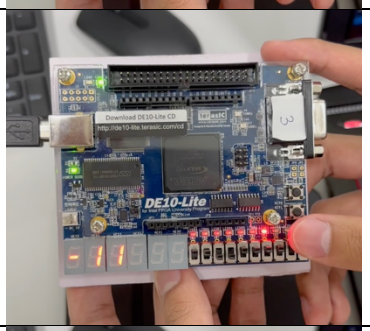
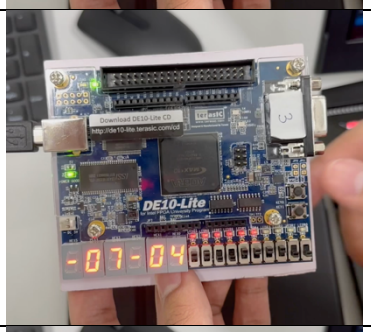
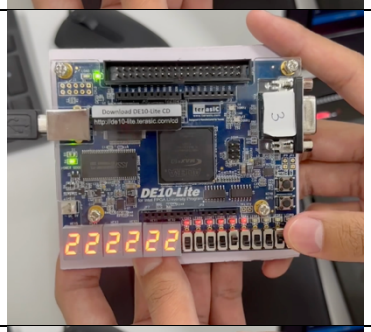
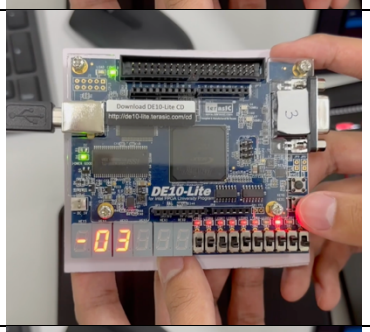
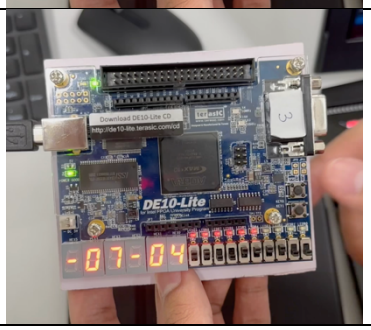
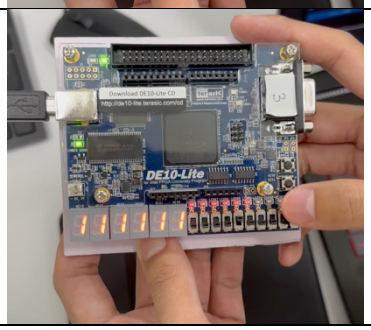
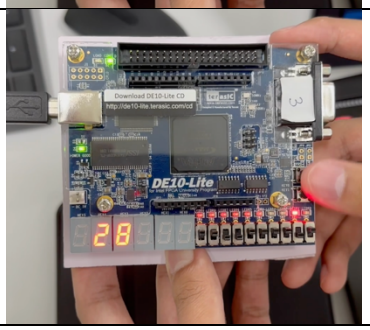
3. เมื่อกดปุ่ม START แล้วปล่อยอีกครั้ง จะเข้าสู่เมนูการแสดงผล โดยแต่ละ Operator จะมีการแสดงผลแตกต่างกันดังนี้
    a. หากเป็นการบวก BCD2 และ BCD1 จะแสดงเลขฐานสิบสองหลัก 1 จำนวน หากเป็นจำนวนลบ BCD3 จะแสดงผลเป็นเครื่องหมายลบ BCD6, BCD5 และ BCD4 จะไม่แสดงผลอะไร หากผลลัพธ์เกินขอบเขตเลข -16 ถึง 15 จะแสดงตัวอักษร “Err” บน BCD3, BCD2 และ BCD1 ตามลำดับ จากนั้น ไฟแสดงสถานะ ณ ตำแหน่ง LEDR3_ADD จะติดขึ้น
    b. หากเป็นการลบ BCD2 และ BCD1 จะแสดงเลขฐานสิบสองหลัก 1 จำนวน หากเป็นจำนวนลบ BCD3 จะแสดงผลเป็นเครื่องหมายลบ BCD6, BCD5 และ BCD4 จะไม่แสดงผลอะไร หากผลลัพธ์เกินขอบเขตเลข -16 ถึง 15 จะแสดงตัวอักษร “Err” บน BCD3, BCD2 และ BCD1 ตามลำดับ จากนั้น ไฟแสดงสถานะ ณ ตำแหน่ง LEDR2_SUB จะติดขึ้น
    c. หากเป็นการคูณ BCD2 และ BCD1 จะแสดงเลขฐานสิบสองหลัก 1 จำนวน หากเป็นจำนวนลบ BCD3 จะแสดงผลเป็นเครื่องหมายลบ BCD6, BCD5 และ BCD4 จะไม่แสดงผลอะไร หากผลลัพธ์เกินขอบเขตเลข -99 ถึง 99 จะแสดงตัวอักษร “Err” บน BCD3, BCD2 และ BCD1 ตามลำดับ จากนั้น ไฟแสดงสถานะ ณ ตำแหน่ง LEDR1_MUL จะติดขึ้น
    d. หากเป็นการหาร BCD2 และ BCD1 จะแสดงเลขฐานสิบสองหลัก 1 จำนวน เป็นค่าผลหาร หากผลหารเป็นจำนวนลบ BCD3 จะแสดงผลเป็นเครื่องหมายลบ BCD5 และ BCD4 จะแสดงเลขฐานสิบสองหลัก 1 จำนวน เป็นค่าเศษจากการหาร หากเศษจากการหารเป็นจำนวนลบ BCD6 จะแสดงผลเป็นเครื่องหมายลบ หากตัวหารที่ผู้ใช้ป้อนเข้าในขั้นตอนที่ 1 เป็น 0 จะแสดงตัวอักษร “Err000” บน BCD6, BCD5, BCD4, BCD3, BCD2 และ BCD1 ตามลำดับ จากนั้น ไฟแสดงสถานะ ณ ตำแหน่ง LEDR0_DIV จะติดขึ้น

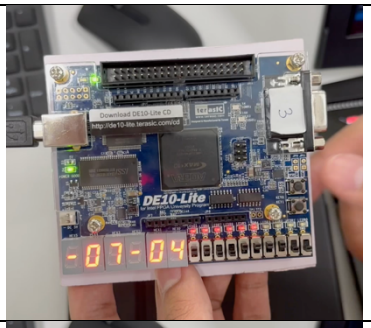
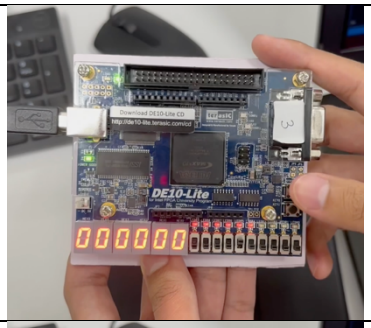
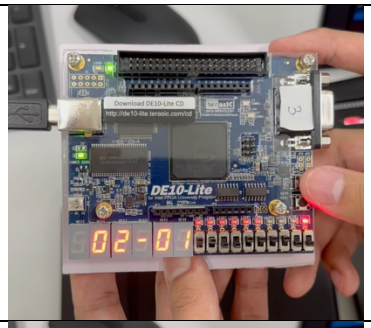
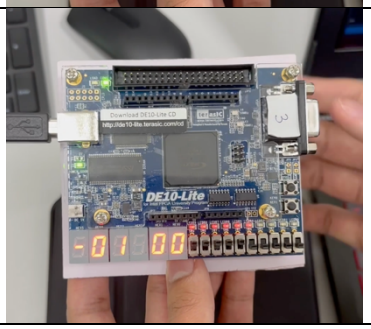
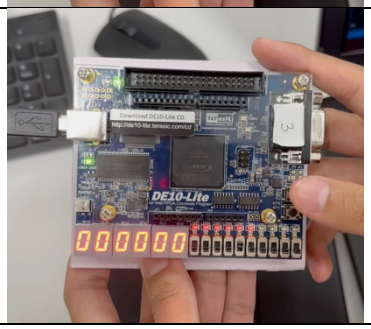
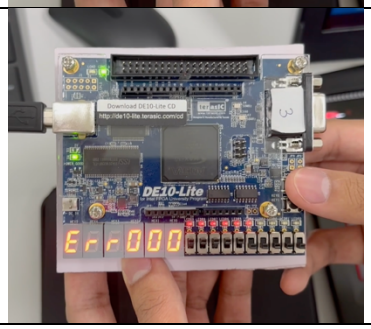
และเมื่อผู้ใช้ต้องการ Reset เพื่อไปสู่ขั้นตอนที่ 1 อีกครั้ง ให้กดปุ่ม RESET 1 ครั้ง เป็นอันเสร็จสิ้นกระบวนการ



ตัวอย่างผลลัพธ์ที่ได้จากการใช้งาน

เงื่อนไข	รูปภาพ
กำหนดให้ A = 01 B = 01 Operator เป็นบวก (3) ผลลัพธ์ = 02	
กำหนดให้ A = 01 B = 01 Operator เป็นลบ (2) ผลลัพธ์ = 00	
กำหนดให้ A = 01 B = 01 Operator เป็นคูณ (1) ผลลัพธ์ = 01	
กำหนดให้ A = 01 B = 01 Operator เป็นหาร (0) ผลลัพธ์ = 01 เศษ 00	
กำหนดให้ A = 13 B = -03 Operator เป็นบวก (3) ผลลัพธ์ = 10	

กำหนดให้ A = 13 B = -03 Operator เป็นลบ (2) ผลลัพธ์ = Err			
กำหนดให้ A = 13 B = -03 Operator เป็นคูณ (1) ผลลัพธ์ = -39			
กำหนดให้ A = 13 B = -03 Operator เป็นหาร (0) ผลลัพธ์ = -04 เศษ 01			
กำหนดให้ A = -07 B = -04 Operator เป็นบวก (3) ผลลัพธ์ = -11			
กำหนดให้ A = -07 B = -04 Operator เป็นลบ (2) ผลลัพธ์ = -03			
กำหนดให้ A = -07 B = -04 Operator เป็นคูณ (1) ผลลัพธ์ = 28			

กำหนดให้ A = -07 B = -04 Operator เป็นหาร (0) ผลลัพธ์ = 02 เศษ -01			
กำหนดให้ A = -01 B = 00 Operator เป็นหาร (0) ผลลัพธ์ = Err000			

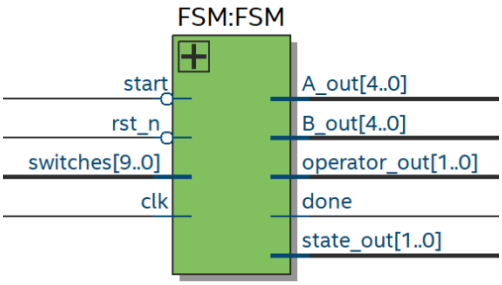
การออกแบบวงจร

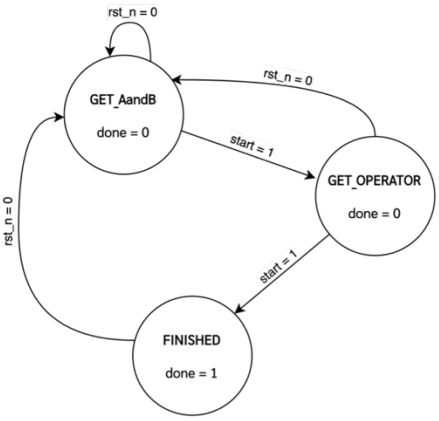
การออกแบบวงจรจะแบ่งออกเป็น 15 ส่วนหลักดังนี้

1. **FSM**: เป็นส่วนที่รับค่า Input มาเป็น Clock, Reset, Start และ Switches อีก 10 ตัวเพื่อรับค่าตัวเลขหรือ Operator ซึ่งหลักการทำงานของวงจรนี้ จะแบ่งออกเป็น 3 State ได้แก่
    a. **Select A and B state :** ใน State นี้ จะรับค่า A และ B ผ่าน Switches ทั้ง 10 ที่รับมาเป็น Input โดย SW5-9 จะเป็นค่า A และ SW0-4 จะเป็นค่า B เมื่อผู้ใช้กดปุ่ม Start หลังจากปรับ Switches ให้เป็นเลขฐานสองจำนวน N-bit ตามที่ต้องการแล้ว ก็จะเก็บค่านั้นไว้ใน A_out และ B_out ของวงจรแล้วกำหนด state_out = "00" และจะเข้าไปสู่ State ถัดไป แต่หากมีการกด Reset จะทำให้ A_out และ B_out ถูก Reset กลับไปเป็นสถานะเดิมและ State จะย้อนกลับมาที่ **Select A and B state** (หรือ State เริ่มต้น)
    b. **Select Operator state :** ใน State นี้ จะรับค่า State ผ่าน Switches 2 ตัวที่รับมาเป็น Input โดยจะเลือกใช้ SW8-9 เป็นตัวกำหนด Operator สามารถเลือกได้ 4 Operators ได้แก่ บวก ลบ คูณ และหาร โดยจะแทนเป็นเลขฐานสอง 2-bit ดังนี้
        i. Operation = “11” เป็นการกระทำฟังก์ชันการบวก A + B
        ii. Operation = “10” เป็นการกระทำฟังก์ชันการลบ A - B
        iii. Operation = “01” เป็นการกระทำฟังก์ชันการคูณ A x B
        iv. Operation = “00” เป็นการกระทำฟังก์ชันการหาร A ÷ B

    เมื่อผู้ใช้กดปุ่ม Start หลังจากปรับ Switches ให้เป็นเลขฐานสอง 2-bit ตามที่ต้องการแล้ว ก็จะเก็บค่า Operator นั้นไว้ใน operator_out แล้วกำหนด state_out = "01" และจะเข้าไปสู่ State ถัดไป แต่หากมีการกด Reset จะทำให้ operator_out ถูก Reset กลับไปเป็นสถานะเดิมและ State จะย้อนกลับไปที่ **Select A and B state** (หรือ State เริ่มต้น)

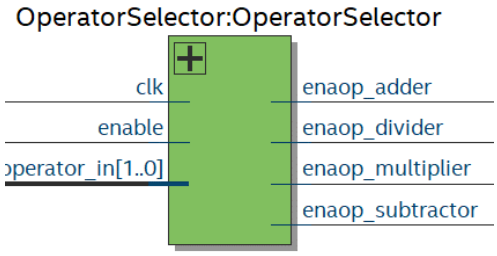
c. **Finished :** ใน State นี้จะมีการกำหนด state_out = "10" และ done = 1 ซึ่งหมายถึงว่าทำการรับ Input ทั้งหมดเรียบร้อย จึงออกจาก FSM ซึ่งค่าที่ส่งออกจะมี A_out กับ B_out ที่เป็น STD_LOGIC_VECTOR จาก N - 1 bit ถึง 0 bit, operator_out กับ state_out ที่เป็น STD_LOGIC_VECTOR จาก 1 ถึง 0 และ done ที่เป็น STD_LOGIC






```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FSM IS
    GENERIC (N : INTEGER := 5); -- 0123
    PORT (
        clk, rst_n, start : IN STD_LOGIC;
        switches : IN STD_LOGIC_VECTOR(2 * N - 1 DOWNTO 0);
        A_out, B_out : OUT STD_LOGIC_VECTOR (N - 1 DOWNTO 0);
        operator_out : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        state_out : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        done : OUT STD_LOGIC
    );
END FSM;

ARCHITECTURE Behavioral OF FSM IS
    TYPE state_type IS (GET_AandB, GET_OPERATOR, FINISHED);
    SIGNAL current_state : state_type := GET_AandB;

BEGIN
    PROCESS (clk, rst_n, start)
    BEGIN
        IF rst_n = '1' THEN
            done <= '0';
            current_state <= GET_AandB;
        ELSIF falling_edge(start) THEN
            CASE current_state IS
                WHEN GET_AandB => current_state <= GET_OPERATOR;
                WHEN GET_OPERATOR => current_state <= FINISHED;
                WHEN FINISHED => current_state <= FINISHED;
            END CASE;
        END IF;

        CASE current_state IS
            WHEN GET_AandB =>
                A_out <= switches(2 * N - 1 DOWNTO (2 * N - 1) - (N - 1));
                B_out <= switches(N - 1 DOWNTO 0);
                state_out <= "00";
            WHEN GET_OPERATOR =>
                operator_out <= switches(1 DOWNTO 0);
                state_out <= "01";
            WHEN FINISHED =>
                done <= '1';
                state_out <= "10";
        END CASE;

    END PROCESS;

END Behavioral;
```

2. **Operator Selector :** เป็นส่วนที่รับค่า Input มาเป็น Clock, Enable และ operator_in ขนาด 2-bit จาก operator_out จากวงจร FSM โดยจะส่งค่าออกเป็น STD_LOGIC โดยมีชื่อดังนี้ enaop_adder, enaop_subtractor, enaop_multiplier, enaop_divider ซึ่งหน้าที่ของแต่ละขาจะเป็นการ Enable วงจร Adder, Subtractor, Multiplier, Divider เพียงอย่างใดอย่างหนึ่ง วงจรนี้มีหน้าที่เลือก Operator ให้ทำงานผ่านขา enable ต่าง ๆ ซึ่งจะไป trigger วงจรคำนวณให้ทำงานตาม operator ที่ผู้ใช้เลือกไว้



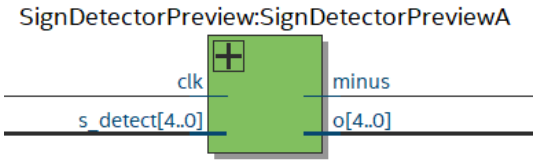

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY OperatorSelector IS
    PORT (
        clk, enable, resetop : IN STD_LOGIC; -- `enable` will derived from the "DONE" signal
        operator_in : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        enaop_adder, enaop_subtractor, enaop_multiplier, enaop_divider : OUT STD_LOGIC -- will be used to enable the adder, subtractor, multiplier and dividerr
    );
END OperatorSelector;

ARCHITECTURE Behavioral OF OperatorSelector IS
BEGIN
    PROCESS (clk, enable, resetop, operator_in)
    BEGIN
        IF rising_edge(clk) THEN
            IF resetop = '1' THEN
                enaop_divider <= '0';
                enaop_subtractor <= '0';
                enaop_multiplier <= '0';
                enaop_adder <= '0';
            ELSIF enable = '1' THEN
                CASE operator_in IS
                    WHEN "00" => -- Division
                        enaop_divider <= '1';
                        enaop_subtractor <= '0';
                        enaop_multiplier <= '0';
                        enaop_adder <= '0';
                    WHEN "01" => -- Multiplication
                        enaop_divider <= '0';
                        enaop_subtractor <= '0';
                        enaop_multiplier <= '1';
                        enaop_adder <= '0';
                    WHEN "10" => -- Subtraction
                        enaop_divider <= '0';
                        enaop_subtractor <= '1';
                        enaop_multiplier <= '0';
                        enaop_adder <= '0';
                    WHEN "11" => -- Addition
                        enaop_divider <= '0';
                        enaop_subtractor <= '0';
                        enaop_multiplier <= '0';
                        enaop_adder <= '1';
                END CASE;
            END IF;
        END IF;
    END PROCESS;
END Behavioral;
```

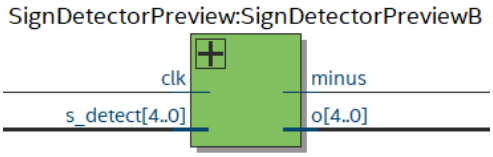
3. **SignDetectorPreview :** เป็นส่วนที่รับค่า Input มาเป็น Clock และ A_out กับ B_out จากวงจร FSM ซึ่งทำหน้าที่ตรวจจับค่าที่เป็นลบแล้วทำ 2's complement และมี Output minus ที่อ้างอิงจาก MSB ของค่า s_detect และเลือก Output ตาม Sign bit

   a. พบ MSB (Sign bit) เป็น 0 ให้ Output o เป็นค่าเดิมที่รับมาจาก s_detect

   b. พบ MSB (Sign bit) เป็น 1 ให้ Output o เป็นค่าของ s_detect ที่ผ่านการทำ 2's complement

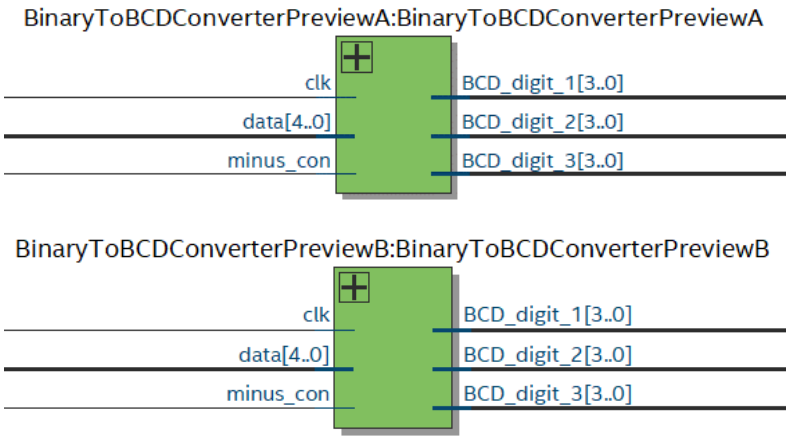
ซึ่งวงจรดังกล่าวนี้จะควบคุมการแสดง Preview ของทั้งค่า A และค่า B





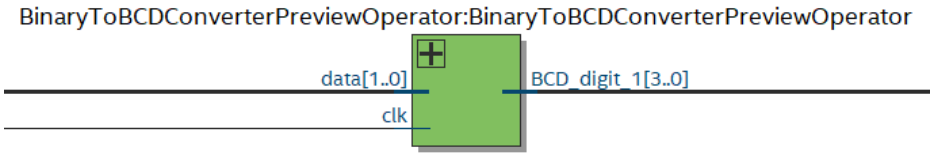

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

ENTITY SignDetectorPreview IS
    GENERIC (N : INTEGER := 5);
    PORT (
        s_detect : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        clk : IN STD_LOGIC;
        minus : OUT STD_LOGIC;
        o : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0));
END SignDetectorPreview;

ARCHITECTURE Structural OF SignDetectorPreview IS
    SIGNAL complemented : STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
    SIGNAL zero : STD_LOGIC_VECTOR(N - 1 DOWNTO 0) := (OTHERS => '0');

BEGIN
    adder : ENTITY work.BinaryAdderAndSubtractor(Structural) GENERIC MAP (N) -- Add 1 to the input
        PORT MAP(
            a => zero, b => s_detect, m => '1', clock => clk, enable => '1', s => complemented -- complemented is the output of the adders
        );

    minus <= s_detect(N - 1);

    WITH s_detect(N - 1) SELECT -- Detect MSB
    o <= s_detect WHEN '0', -- If MSB is 0, output is s_detect
        complemented WHEN OTHERS; -- If MSB is 1, output is complemented
END ARCHITECTURE;
```

4. **BinaryToBCDConverterPreview :** เป็นส่วนที่รับค่า Input มาเป็น Clock, minus_con ที่รับค่ามาจาก minus และ data ที่รับค่ามาจาก o ของ **SignDetectorPreview** ซึ่งในวงจรนี้จะเป็นการแปลง data ที่เป็น STD_LOGIC_VECTOR N-bit เป็นจำนวน Integer ในแต่ละหลัก ซึ่งใน Component นี้จะแบ่งเป็น
    a. BCD_digit_1, BCD_digit_2 ที่เป็นการแปลง Integer เป็น STD_LOGIC_VECTOR ขนาด N-1 bit
    b. เมื่อ minus_con = '1' จะทำให้ BCD_digit_3 = "1011" ซึ่งคือเครื่องหมายลบ (Segment g)
    c. เมื่อ minus_con = '0' จะทำให้ BCD_digit_3 = "1010" ซึ่งคือการไม่แสดงผลใด ๆ บน 7-Segment หลักที่ 3




```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.STD_LOGIC_ARITH.ALL;

ENTITY BinaryToBCDConverterPreviewA IS
    GENERIC (
        N : INTEGER := 5
    );
    PORT (
        clk : IN STD_LOGIC;
        minus_con : IN STD_LOGIC;
        data : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        BCD_digit_1 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0)
    );
END BinaryToBCDConverterPreviewA;

ARCHITECTURE Structural OF BinaryToBCDConverterPreviewA IS
    SIGNAL signal_integer1 : INTEGER := 0;
    SIGNAL signal_integer2 : INTEGER := 0;
BEGIN
    PROCESS (clk)
    BEGIN
        IF rising_edge(clk) THEN
            signal_integer1 <= conv_integer(unsigned(data)) MOD 10;
            signal_integer2 <= (conv_integer(unsigned(data)) / 10) MOD 10;

            BCD_digit_1 <= conv_std_logic_vector(signal_integer1, N - 1);
            BCD_digit_2 <= conv_std_logic_vector(signal_integer2, N - 1);

            IF (minus_con = '1') THEN -- if MSB is 1
                BCD_digit_3 <= "1011"; -- minus
            ELSE
                BCD_digit_3 <= "1010"; -- none
            END IF;
        END IF;
    END PROCESS;
END Structural;
```

5. **BinaryToBCDConverterPreviewOperator :** เป็นส่วนรับค่า Input เป็น data จาก operator_out ของ FSM และเป็น STD_LOGIC_VECTOR ขนาด 2-bit หน้าที่คือการเพิ่ม bit ให้กับค่า operator_out ที่มาจาก FSM และส่งค่าออกให้กับ BCDto7Segment เพื่อนำไปแสดงผลต่อไป




```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY BinaryToBCDConverterPreviewOperator IS
5      PORT (
6          clk : IN STD_LOGIC;
7          data : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
8          BCD_digit_1 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
9      );
10 END BinaryToBCDConverterPreviewOperator;
11
12 ARCHITECTURE Structural OF BinaryToBCDConverterPreviewOperator IS
13     SIGNAL signal_integer : INTEGER := 0;
14 BEGIN
15     PROCESS (clk)
16     BEGIN
17         IF rising_edge(clk) THEN
18             BCD_digit_1 <= "00" & data;
19         END IF;
20     END PROCESS;
21 END Structural;
```

6. **MUX18to3_Result :** เป็นส่วนที่ควบคุมการเลือกแสดงค่าบน 7-Segment โดยขา control จะรับจากขา state_out ขนาด 2 bit ที่มาจาก FSM

   เมื่อ control = 00 และ control = 01 จะแสดงค่า Preview ตาม **Select A and B state** (เฉพาะส่วนตัวเลข A) **กับ Select Operator state** และจะแสดงค่า Result ใน Finished state เมื่อ control = 10 ซึ่งจะแบ่งเป็นเงื่อนไขดังนี้

   i. Operation = “11” เป็นการกระทำฟังก์ชันการบวก A + B

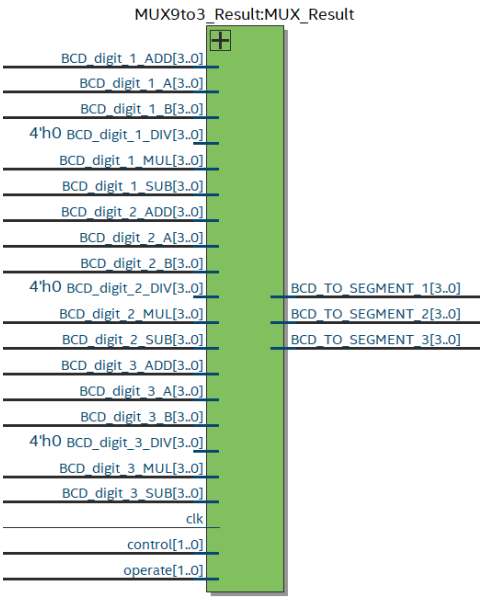
   ii. Operation = “10” เป็นการกระทำฟังก์ชันการลบ A - B

   iii. Operation = “01” เป็นการกระทำฟังก์ชันการคูณ A x B

   iv. Operation = “00” เป็นการกระทำฟังก์ชันการหาร A ÷ B

และค่า Output จะออกเป็น BCD ขนาด N-2 bit จำนวน 3 หลัก เพื่อจะนำไปเชื่อมต่อกับ BCD to 7-Segment จำนวน 3 หลัก

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MUX18to3_Result IS
    GENERIC (N : INTEGER := 5);
    PORT (
        BCD_digit_1_A : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2_A : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3_A : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_1_B : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2_B : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3_B : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_digit_1_ADD : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2_ADD : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3_ADD : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_digit_1_SUB : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2_SUB : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3_SUB : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_digit_1_MUL : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2_MUL : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3_MUL : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_digit_1_DIV : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2_DIV : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3_DIV : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_TO_SEGMENT_1 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_TO_SEGMENT_2 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_TO_SEGMENT_3 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        DONE_LED_ADD : OUT STD_LOGIC;
        DONE_LED_SUB : OUT STD_LOGIC;

        control : IN STD_LOGIC_VECTOR;
        operate : IN STD_LOGIC_VECTOR;
        clk : IN STD_LOGIC
    );
END MUX18to3_Result;

ARCHITECTURE Behavioral OF MUX18to3_Result IS
    SIGNAL BCD_TO_SEGMENT_1_temp : STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
    SIGNAL BCD_TO_SEGMENT_2_temp : STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
    SIGNAL BCD_TO_SEGMENT_3_temp : STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
    SIGNAL DONE_LED_ADD_temp : STD_LOGIC := '0';
    SIGNAL DONE_LED_SUB_temp : STD_LOGIC := '0';
BEGIN
    PROCESS (clk)
    BEGIN

        IF rising_edge(clk) THEN
            IF control = "00" THEN
                BCD_TO_SEGMENT_1_temp <= BCD_digit_1_A;
                BCD_TO_SEGMENT_2_temp <= BCD_digit_2_A;
                BCD_TO_SEGMENT_3_temp <= BCD_digit_3_A;
                DONE_LED_ADD_temp <= '0';
                DONE_LED_SUB_temp <= '0';
            ELSIF control = "01" THEN
                BCD_TO_SEGMENT_1_temp <= BCD_digit_1_B;
                BCD_TO_SEGMENT_2_temp <= BCD_digit_2_B;
                BCD_TO_SEGMENT_3_temp <= BCD_digit_3_B;
                DONE_LED_ADD_temp <= '0';
                DONE_LED_SUB_temp <= '0';
            ELSIF control = "10" THEN

                IF operate = "11" THEN
                    DONE_LED_ADD_temp <= '1';
                ELSIF operate = "10" THEN
                    DONE_LED_SUB_temp <= '1';
                END IF;

                IF operate = "00" THEN
                    BCD_TO_SEGMENT_1_temp <= BCD_digit_1_DIV;
                    BCD_TO_SEGMENT_2_temp <= BCD_digit_2_DIV;
                    BCD_TO_SEGMENT_3_temp <= BCD_digit_3_DIV;
                ELSIF operate = "01" THEN
                    BCD_TO_SEGMENT_1_temp <= BCD_digit_1_MUL;
                    BCD_TO_SEGMENT_2_temp <= BCD_digit_2_MUL;
                    BCD_TO_SEGMENT_3_temp <= BCD_digit_3_MUL;
                ELSIF operate = "10" THEN
                    BCD_TO_SEGMENT_1_temp <= BCD_digit_1_SUB;
                    BCD_TO_SEGMENT_2_temp <= BCD_digit_2_SUB;
                    BCD_TO_SEGMENT_3_temp <= BCD_digit_3_SUB;
                ELSIF operate = "11" THEN
                    BCD_TO_SEGMENT_1_temp <= BCD_digit_1_ADD;
                    BCD_TO_SEGMENT_2_temp <= BCD_digit_2_ADD;
                    BCD_TO_SEGMENT_3_temp <= BCD_digit_3_ADD;
                END IF;
            END IF;
        END IF;
    END PROCESS;

    BCD_TO_SEGMENT_1 <= BCD_TO_SEGMENT_1_temp;
    BCD_TO_SEGMENT_2 <= BCD_TO_SEGMENT_2_temp;
    BCD_TO_SEGMENT_3 <= BCD_TO_SEGMENT_3_temp;
    DONE_LED_ADD <= DONE_LED_ADD_temp;
    DONE_LED_SUB <= DONE_LED_SUB_temp;
END Behavioral;
```



7. **MUX18to3_Remainder :** เป็นส่วนที่ควบคุมการเลือกแสดงค่าบน 7-Segment โดยขา control จะรับจากขา state_out ขนาด 2 bit ที่มาจาก FSM

เมื่อ control = 00 และ control = 01 จะแสดงค่า Preview ตาม **Select A and B state** (เฉพาะส่วนตัวเลข B) **กับ Select Operator state** และจะแสดงค่า Result ใน Finished state เมื่อ control = 10 ซึ่งจะแบ่งเป็นเงื่อนไขดังนี้

i. Operation = “11” เป็นการกระทำฟังก์ชันการบวก A + B (ออกมาเป็นเครื่องหมาย - - -)

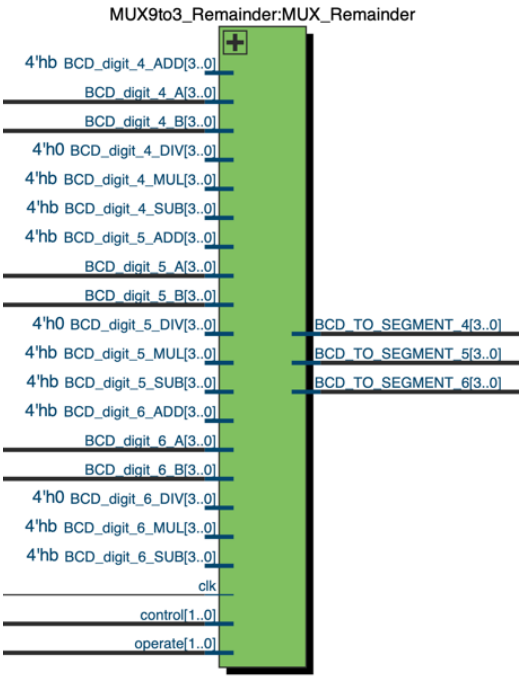
ii. Operation = “10” เป็นการกระทำฟังก์ชันการลบ A – B (ออกมาเป็นเครื่องหมาย - - -)

iii. Operation = “01” เป็นการกระทำฟังก์ชันการคูณ A x B (ออกมาเป็นเครื่องหมาย - - -)

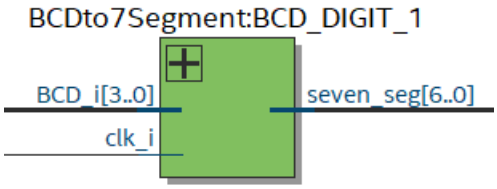
iv. Operation = “00” เป็นการกระทำฟังก์ชันการหาร A ÷ B

และค่า Output จะออกเป็น BCD ขนาด N-2 bit จำนวน 3 หลัก เพื่อจะนำไปเชื่อมต่อกับ BCD to 7-Segment จำนวน 3 หลัก สำหรับฟังก์ชั่นการบวก ลบ และคูณ จะแสดงค่า BCD ออกมาเป็นค่า ‘1011’ โดยเมื่อนำไปเชื่อมต่อกับ BCD to 7-Segment จำนวน 3 หลัก จะแสดงออกมาเป็น ‘---'

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MUX18to3_Remainder IS
    GENERIC (N : INTEGER := 5);
    PORT (
        BCD_digit_4_A : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_5_A : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_6_A : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_4_B : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_5_B : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_6_B : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_digit_4_ADD : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_5_ADD : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_6_ADD : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_digit_4_SUB : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_5_SUB : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_6_SUB : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_digit_4_MUL : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_5_MUL : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_6_MUL : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_digit_4_DIV : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_5_DIV : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_6_DIV : IN STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        BCD_TO_SEGMENT_4 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_TO_SEGMENT_5 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_TO_SEGMENT_6 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);

        control : IN STD_LOGIC_VECTOR;
        operate : IN STD_LOGIC_VECTOR;
        clk : IN STD_LOGIC
    );
END MUX18to3_Remainder;

ARCHITECTURE Behavioral OF MUX18to3_Remainder IS
    SIGNAL BCD_TO_SEGMENT_4_temp : STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
    SIGNAL BCD_TO_SEGMENT_5_temp : STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
    SIGNAL BCD_TO_SEGMENT_6_temp : STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
BEGIN
    PROCESS (clk)
    BEGIN
        IF rising_edge(clk) THEN
            IF control = "00" THEN
                BCD_TO_SEGMENT_4_temp <= BCD_digit_4_A;
                BCD_TO_SEGMENT_5_temp <= BCD_digit_5_A;
                BCD_TO_SEGMENT_6_temp <= BCD_digit_6_A;
            ELSIF control = "01" THEN
                BCD_TO_SEGMENT_4_temp <= BCD_digit_4_B;
                BCD_TO_SEGMENT_5_temp <= BCD_digit_5_B;
                BCD_TO_SEGMENT_6_temp <= BCD_digit_6_B;
            ELSIF control = "10" THEN
                IF operate = "00" THEN
                    BCD_TO_SEGMENT_4_temp <= BCD_digit_4_DIV;
                    BCD_TO_SEGMENT_5_temp <= BCD_digit_5_DIV;
                    BCD_TO_SEGMENT_6_temp <= BCD_digit_6_DIV;
                ELSIF operate = "01" THEN
                    BCD_TO_SEGMENT_4_temp <= BCD_digit_4_MUL;
                    BCD_TO_SEGMENT_5_temp <= BCD_digit_5_MUL;
                    BCD_TO_SEGMENT_6_temp <= BCD_digit_6_MUL;
                ELSIF operate = "10" THEN
                    BCD_TO_SEGMENT_4_temp <= BCD_digit_4_SUB;
                    BCD_TO_SEGMENT_5_temp <= BCD_digit_5_SUB;
                    BCD_TO_SEGMENT_6_temp <= BCD_digit_6_SUB;
                ELSIF operate = "11" THEN
                    BCD_TO_SEGMENT_4_temp <= BCD_digit_4_ADD;
                    BCD_TO_SEGMENT_5_temp <= BCD_digit_5_ADD;
                    BCD_TO_SEGMENT_6_temp <= BCD_digit_6_ADD;
                END IF;
            END IF;
        END IF;
    END PROCESS;

    BCD_TO_SEGMENT_4 <= BCD_TO_SEGMENT_4_temp;
    BCD_TO_SEGMENT_5 <= BCD_TO_SEGMENT_5_temp;
    BCD_TO_SEGMENT_6 <= BCD_TO_SEGMENT_6_temp;
END Behavioral;
```

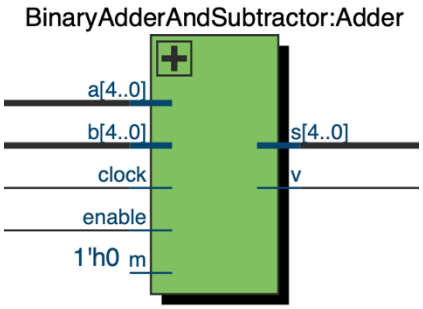


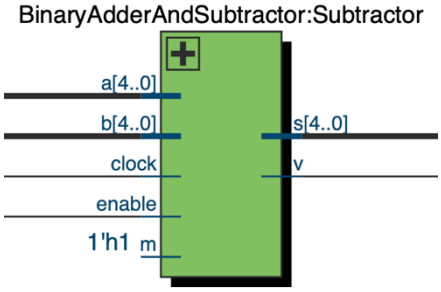
8. **BCDto7Segment :** เป็นส่วนที่แปลง BCD เป็น 7-Segment ที่เป็น STD_LOGIC_VECTOR ขนาด 7 bit โดยจะเลือกกรณีการปล่อยสัญญาณ 7 Segment จาก BCD_i หากเป็นค่าเท่าไหร่ ก็จะแสดงค่าออกมาเป็นเลขนั้น




```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY BCDto7Segment IS
    GENERIC (N : INTEGER := 5); -- N is the number of bits of the input
    PORT (
        BCD_i : IN STD_LOGIC_VECTOR (N - 2 DOWNTO 0);
        clk_i : IN STD_LOGIC;
        seven_seg : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)); -- gfedcba
END BCDto7Segment;

ARCHITECTURE data_process OF BCDto7Segment IS -- data_process is the name of the architectur
BEGIN
    PROCESS (clk_i) -- sensitivity list
    BEGIN
        IF clk_i'event AND clk_i = '1' THEN
            CASE BCD_i IS --gfedcba (This 7-segment is active low, common anode)
                WHEN "0000" => seven_seg <= "1000000"; --7-segment display number 0
                WHEN "0001" => seven_seg <= "1111001"; --7-segment display number 1
                WHEN "0010" => seven_seg <= "0100100"; --7-segment display number 2
                WHEN "0011" => seven_seg <= "0110000"; --7-segment display number 3
                WHEN "0100" => seven_seg <= "0011001"; --7-segment display number 4
                WHEN "0101" => seven_seg <= "0010010"; --7-segment display number 5
                WHEN "0110" => seven_seg <= "0000010"; --7-segment display number 6
                WHEN "0111" => seven_seg <= "1111000"; --7-segment display number 7
                WHEN "1000" => seven_seg <= "0000000"; --7-segment display number 8
                WHEN "1001" => seven_seg <= "0010000"; --7-segment display number 9

                WHEN "1011" => seven_seg <= "0111111"; --7-segment display minus

                WHEN "1100" => seven_seg <= "0000110"; --7-segment display E
                WHEN "1101" => seven_seg <= "0101111"; --7-segment display r

                WHEN OTHERS => seven_seg <= "1111111"; --7-segment display none
            END CASE;
        END IF;
    END PROCESS;
END data_process;
```

9. **BinaryAdder และ BinarySubtractor :** เป็นส่วนที่รับค่า Input มาเป็น Clock, Enable และ A กับ B ขนาด N-bit จาก A out และ B out ที่เก็บค่าจากวงจร FSM โดยจะส่งค่าออกเป็น STD_LOGIC_VECTOR เป็นค่า sum ขนาด N-bit ที่ได้จากการคำนวณและ STD_LOGIC เป็นค่า V ซึ่งเป็นขา overflow โดยจะส่งค่านี้ไปที่ BinaryToBCDConverter เพื่อประมวลการแสดงผลต่อไป หน้าที่ของวงจรนี้คือการบวกหรือลบจำนวน 2 จำนวนด้วยวงจร Binary adder and subtractor โดยหากเป็นการบวก กำหนดขา m เป็น 0 และหากเป็นการลบ กำหนดขา m เป็น 1

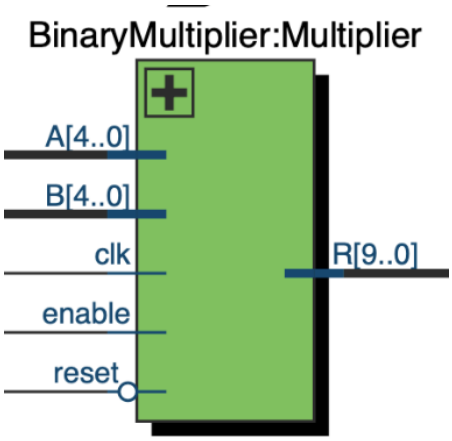




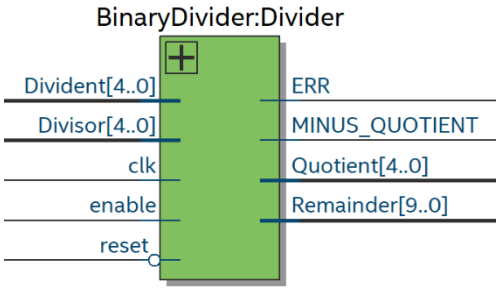

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.STD_LOGIC_ARITH.ALL;
USE ieee.STD_LOGIC_UNSIGNED.ALL;

ENTITY BinaryAdderAndSubtractor IS
    GENERIC (N : INTEGER := 5);
    PORT (
        a, b : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        m, clock, enable : IN STD_LOGIC;
        s : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        c : OUT STD_LOGIC_VECTOR(N DOWNTO 1);
        v : OUT STD_LOGIC
    );
END BinaryAdderAndSubtractor;

ARCHITECTURE Structural OF BinaryAdderAndSubtractor IS
    COMPONENT FullAdder IS
        PORT (
            x, y, z, clk, enable_fulladder : IN STD_LOGIC;
            s, c : OUT STD_LOGIC
        );
    END COMPONENT;
    SIGNAL c_internal : STD_LOGIC_VECTOR(1 TO N);
BEGIN
    FA0 : FullAdder
    PORT MAP(
        x => a(0),
        y => m XOR b(0),
        z => m,
        clk => clock,
        enable_fulladder => enable,
        s => s(0),
        c => c_internal(1)
    );

    FA_gen : FOR i IN 1 TO N - 1 GENERATE
        FAi : FullAdder
        PORT MAP(
            x => a(i),
            y => m XOR b(i),
            z => c_internal(i),
            clk => clock,
            enable_fulladder => enable,
            s => s(i),
            c => c_internal(i + 1)
        );
    END GENERATE;

    v <= c_internal(N) XOR c_internal(N - 1); -- Calculate overflow

END Structural;
```

10. **BinaryMultiplier :** เป็นส่วนที่รับค่า Input มาเป็น Clock, Enable, Reset และ A กับ B ขนาด N-bit จาก A out และ B out ที่เก็บค่าจากวงจร FSM โดยจะส่งค่าออกเป็น STD_LOGIC_VECTOR เป็นค่า result ขนาด 2*N-bit ที่ได้จากการคำนวณ โดยจะส่งค่านี้ไปที่ BinaryToBCDConverter เพื่อประมวลการแสดงผลต่อไป หน้าที่ของวงจรนี้คือการคูณจำนวน 2 จำนวน หากค่า A หรือ B ที่ได้มาค่าใดค่าหนึ่งมี MSB = 1 จะทำ 2 compliment ก่อนที่จะเข้าสู่กระบวนการคูณ

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY BinaryMultiplier IS
    GENERIC (N : INTEGER := 5);
    PORT (
        clk : IN STD_LOGIC;
        enable : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        A, B : IN STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
        R : OUT STD_LOGIC_VECTOR (2 * N - 1 DOWNTO 0) := (OTHERS => '0');
        DONE : OUT STD_LOGIC := '0');
END BinaryMultiplier;

ARCHITECTURE Behavioral OF BinaryMultiplier IS
    TYPE state_type IS (s0, s1, s2);
    SIGNAL state : state_type := s0;
    SIGNAL s_start : STD_LOGIC := '1';
    SIGNAL normalised_data_A : STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL normalised_data_B : STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL data_A : STD_LOGIC_VECTOR (2 * N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL data_B : STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL data_R : STD_LOGIC_VECTOR (2 * N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL counter : INTEGER := 0;
BEGIN
    PROCESS (clk, reset, enable)
    BEGIN
        IF reset = '1' THEN
            -- toggle reset
            state <= s0;
            s_start <= '1';
            DONE <= '0';
            counter <= 0;
            R <= (OTHERS => '0');
            data_A <= (OTHERS => '0');
            data_B <= (OTHERS => '0');
            data_R <= (OTHERS => '0');

        ELSIF rising_edge(clk) THEN

            -- detect a sign bit. if it's 1, do 2 compliment.
            IF A(N - 1) = '1' THEN
                normalised_data_A <= NOT A + 1;
            END IF;
            IF B(N - 1) = '1' THEN
                normalised_data_B <= NOT B + 1;
            END IF;
            IF A(N - 1) = '0' THEN
                normalised_data_A <= A;
            END IF;
            IF B(N - 1) = '0' THEN
                normalised_data_B <= B;
            END IF;

            -- FSM for BinaryMultiplier
            CASE state IS
                WHEN s0 =>
                    IF enable = '1' AND s_start = '1' THEN
                        data_A (N - 1 DOWNTO 0) <= normalised_data_A;--normalised_data_A;
                        data_B <= normalised_data_B;--normalised_data_B;
                        state <= s1;
                    ELSE
                        state <= s0;
                    END IF;

                WHEN s1 =>
                    IF (counter <= N) THEN
                        state <= s1;
                        IF data_B(counter) = '1' THEN
                            data_R <= data_R + data_A;
                            data_A <= STD_LOGIC_VECTOR(shift_left(unsigned(data_A), 1));
                            R <= data_R;
                            counter <= counter + 1;
                        ELSE
                            data_A <= STD_LOGIC_VECTOR(shift_left(unsigned(data_A), 1));
                            R <= data_R;
                            counter <= counter + 1;
                        END IF;
                    ELSE
                        state <= s2;
                    END IF;
                WHEN OTHERS =>
                    DONE <= '1';
                    --state <= s0;
            END CASE;
        END IF;
    END PROCESS;
END Behavioral;
```



11. **BinaryDivider :** เป็นส่วนที่รับค่า Input มาเป็น Clock, Enable, Reset และ A กับ B ขนาด N-bit จาก A out และ B out ที่เก็บค่าจากวงจร FSM โดยจะส่งค่าออกเป็น STD_LOGIC_VECTOR เป็นค่า Quotient ขนาด N-bit, STD_LOGIC_VECTOR เป็นค่า Remainder ขนาด 2*N-bit, STD_LOGIC เป็นค่า MINUS_QUOTIENT และ STD_LOGIC เป็นค่า MINUS_REMAINDER ที่ได้จากการคำนวณ โดยการแสดงค่าผลการหารว่าจะกำหนดให้เป็นบวกหรือเป็นลบก็จะแตกต่างกันตามค่า MINUS_QUOTIENT และ MINUS_REMAINDER ซึ่งถูกกำหนดโดย Sign bit ของ Input (แทนบวกเป็น 0 แทนลบเป็น 1) เช่น
    a. หาก A เป็น + และ B เป็น + ให้กำหนดค่า MINUS_QUOTIENT เป็น 0 และ MINUS_REMAINDER เป็น 0
    b. หาก A เป็น - และ B เป็น + ให้กำหนดค่า MINUS_QUOTIENT เป็น 1 และ MINUS_REMAINDER เป็น 0
    c. หาก A เป็น + และ B เป็น - ให้กำหนดค่า MINUS_QUOTIENT เป็น 1 และ MINUS_REMAINDER เป็น 1
    d. หาก A เป็น - และ B เป็น - ให้กำหนดค่า MINUS_QUOTIENT เป็น 0 และ MINUS_REMAINDER เป็น 1

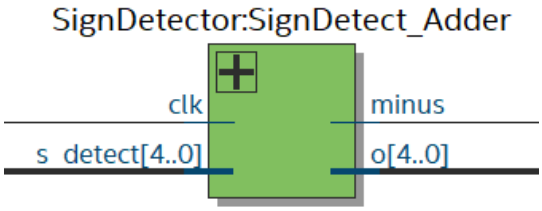


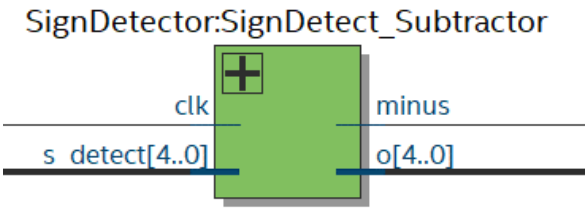
```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY BinaryDivider IS
    GENERIC (N : INTEGER := 5);
    PORT (
        clk : IN STD_LOGIC;
        enable : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        Divident, Divisor : IN STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
        Quotient : OUT STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
        Remainder : OUT STD_LOGIC_VECTOR (2 * N - 1 DOWNTO 0) := (OTHERS => '0');
        MINUS_QUOTIENT : OUT STD_LOGIC;
        MINUS_REMAINDER : OUT STD_LOGIC;
        ERR : OUT STD_LOGIC;
        DONE : OUT STD_LOGIC := '0');
END BinaryDivider;

ARCHITECTURE Behavioral OF BinaryDivider IS
    TYPE state_type IS (s0, s1, s2);
    SIGNAL state : state_type := s0;
    SIGNAL s_start : STD_LOGIC := '1';
    SIGNAL normalised_data_Dividend : STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL normalised_data_Divisor : STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL data_Divisor : STD_LOGIC_VECTOR (2 * N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL data_Quotient : STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL data_Remainder : STD_LOGIC_VECTOR (2 * N - 1 DOWNTO 0) := (OTHERS => '0');
    SIGNAL counter : INTEGER := 0;
BEGIN

    PROCESS (clk, reset, enable)
    BEGIN
        IF Divisor = "00000" THEN
            ERR <= '1';
        ELSE
            ERR <= '0';
            IF reset = '1' THEN
                -- toggle reset
                state <= s0;
                s_start <= '1';
                counter <= 0;
                DONE <= '0';
                Quotient <= (OTHERS => '0');
                Remainder <= (OTHERS => '0');
                data_Divisor <= (OTHERS => '0');
                data_Quotient <= (OTHERS => '0');
                data_Remainder <= (OTHERS => '0');
            ELSIF rising_edge(clk) THEN

                -- detect a sign bit. if it's 1, do 2 compliment.
                IF Divident(N - 1) = '1' THEN
                    normalised_data_Dividend <= NOT Divident + 1;
                END IF;
                IF Divisor(N - 1) = '1' THEN
                    normalised_data_Divisor <= NOT Divisor + 1;
                END IF;
                IF Divident(N - 1) = '0' THEN
                    normalised_data_Dividend <= Divident;
                END IF;
                IF Divisor(N - 1) = '0' THEN
                    normalised_data_Divisor <= Divisor;
                END IF;

                -- FSM for BinaryDivider
                CASE state IS
                    WHEN s0 =>
                        IF enable = '1' AND s_start = '1' THEN
                            data_Divisor <= STD_LOGIC_VECTOR(normalised_data_Divisor) & STD_LOGIC_VECTOR(to_unsigned(0, N));
                            data_Remainder <= STD_LOGIC_VECTOR(to_unsigned(0, N)) & STD_LOGIC_VECTOR(normalised_data_Dividend);
                            state <= s1;
                        ELSE
                            state <= s0;
                            DONE <= '0';
                        END IF;

                    WHEN s1 =>
                        IF (counter <= N) THEN
                            state <= s1;
                            IF data_Remainder < data_Divisor THEN
                                data_Divisor <= STD_LOGIC_VECTOR(shift_right(unsigned(data_Divisor), 1));
                                data_Quotient <= STD_LOGIC_VECTOR(shift_left(unsigned(data_Quotient), 1));
                                counter <= counter + 1;
                            ELSIF data_Remainder >= data_Divisor THEN
                                data_Remainder <= STD_LOGIC_VECTOR(unsigned(data_Remainder) - unsigned(data_Divisor));
                                data_Divisor <= STD_LOGIC_VECTOR(shift_right(unsigned(data_Divisor), 1));
                                data_Quotient <= STD_LOGIC_VECTOR(data_Quotient(N - 2 DOWNTO 0)) & "1";
                                counter <= counter + 1;
                            END IF;
                        ELSE
                            IF Divident(N - 1) = '0' AND Divisor(N - 1) = '0' THEN
                                Quotient <= data_Quotient;
                                Remainder <= data_Remainder;
                                MINUS_QUOTIENT <= '0';
                                MINUS_REMAINDER <= '0';
                            ELSIF Divident(N - 1) = '1' AND Divisor(N - 1) = '0' THEN
                                Quotient <= data_Quotient;
                                Remainder <= data_Remainder;
                                MINUS_QUOTIENT <= '1';
                                MINUS_REMAINDER <= '0';
                            ELSIF Divident(N - 1) = '0' AND Divisor(N - 1) = '1' THEN
                                Quotient <= data_Quotient;
                                Remainder <= data_Remainder;
                                MINUS_QUOTIENT <= '1';
                                MINUS_REMAINDER <= '0';
                            ELSIF Divident(N - 1) = '1' AND Divisor(N - 1) = '1' THEN
                                Quotient <= data_Quotient;
                                Remainder <= data_Remainder;
                                MINUS_QUOTIENT <= '0';
                                MINUS_REMAINDER <= '1';
                            END IF;
                            state <= s2;
                        END IF;
                    WHEN OTHERS =>
                        DONE <= '1';
                END CASE;
            END IF;
        END IF;
    END PROCESS;
END Behavioral;
```

12. **SignDetectorAdder และ SignDetectorSubtractor :** เป็นส่วนที่รับค่ามาจากขา s ของ **BinaryAdder** กับ **BinarySubtractor** ซึ่งต่อเข้ากับขา s_detector ซึ่งทั้งคู่เป็น STD_LOGIC_VECTOR ขนาด N-bit และเลือก Output ตาม Sign bit

    a. พบ MSB เป็น 0 ให้ Output o เป็นค่าเดิมที่รับมาจาก s_detect

    b. พบ MSB เป็น 1 ให้ Output o เป็นค่าของ s_detect ที่ผ่านการทำ 2's complement





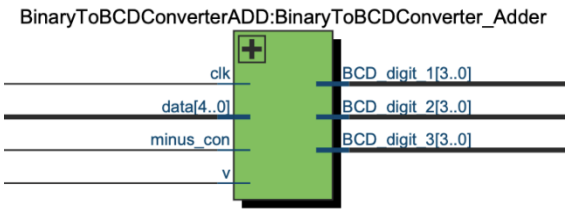

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.std_logic_signed.ALL;
4
5  ENTITY SignDetector IS
6      GENERIC (N : INTEGER := 5);
7      PORT (
8          s_detect : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
9          clk : IN STD_LOGIC;
10         minus : OUT STD_LOGIC;
11         o : OUT STD_LOGIC_VECTOR(N - 1 DOWNTO 0));
12 END SignDetector;
13
14 ARCHITECTURE Structural OF SignDetector IS
15     SIGNAL complemented : STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
16     SIGNAL plusone : STD_LOGIC_VECTOR(N - 1 DOWNTO 0) := (0 => '1', OTHERS => '0');
17
18 BEGIN
19     -- Add 1 to the input
20     adder : ENTITY work.BinaryAdderAndSubtractor(Structural) -- Add 1 to the input
21         PORT MAP(
22             a => NOT s_detect,
23             b => plusone,
24             m => '0',
25             clock => clk,
26             enable => '1',
27             s => complemented -- complemented is the output of the adders
28         );
29
30     minus <= s_detect(N - 1);
31
32     WITH s_detect(N - 1) SELECT -- Detect MSB
33     o <= s_detect WHEN '0', -- If MSB is 0, output is s_detect
34         complemented WHEN OTHERS; -- If MSB is 1, output is complemented
35
36 END Structural;
```

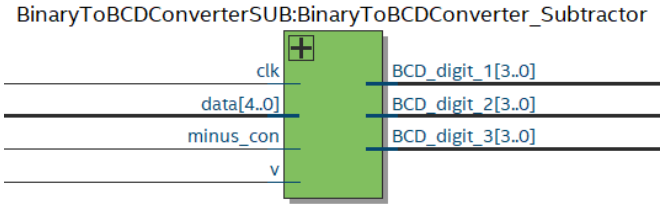
13. **BinaryToBCDConverterAdder และ BinaryToBCDConverterSubtractor :** เป็นส่วนที่รับค่า Input มาเป็น Clock, minus_con ที่รับค่ามาจาก minus และ data ที่รับค่ามาจาก o ของ SignDetector_Adder และ SignDetector_Subtractor และมีขา Input V ที่เป็นขาสำหรับรับค่า Overflow จาก Sign Detector ของ Adder และ Subtractor โดยในวงจรนี้จะเป็นการแปลง data ที่เป็น STD_LOGIC_VECTOR N-bit เป็นจำนวน Integer ในแต่ละหลัก ซึ่งใน Component นี้จะแบ่งเป็น

    a. BCD_digit_1, BCD_digit_2 ที่เป็นการแปลง Integer เป็น STD_LOGIC_VECTOR ขนาด N-1 bit

    b. เมื่อ minus_con = '1' จะทำให้ BCD_digit_3 = "1011" ซึ่งคือเครื่องหมายลบ (Segment g)

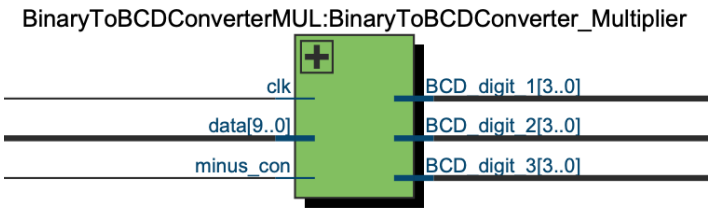
    c. เมื่อ minus_con = '0' จะทำให้ BCD_digit_3 = "1010" ซึ่งคือการไม่แสดงผลใด ๆ บน 7-Segment หลักที่ 3





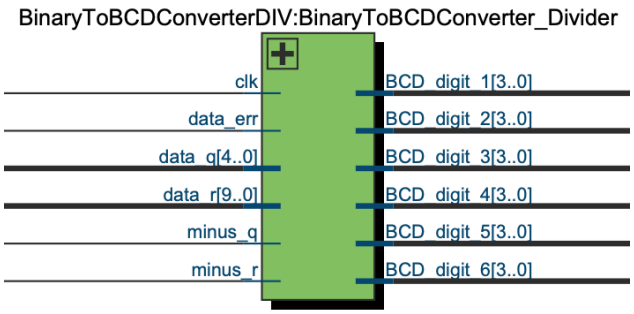

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.STD_LOGIC_ARITH.ALL;

ENTITY BinaryToBCDConverterADD IS
    GENERIC (
        N : INTEGER := 5
    );
    PORT (
        clk : IN STD_LOGIC;
        v : IN STD_LOGIC;
        minus_con : IN STD_LOGIC;
        data : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        BCD_digit_1 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0)
    );
END BinaryToBCDConverterADD;

ARCHITECTURE Structural OF BinaryToBCDConverterADD IS
    SIGNAL signal_integer1 : INTEGER := 0;
    SIGNAL signal_integer2 : INTEGER := 0;
BEGIN
    PROCESS (clk)
    BEGIN
        IF rising_edge(clk) THEN
            IF v = '1' THEN
                BCD_digit_1 <= "1101";
                BCD_digit_2 <= "1101";
                BCD_digit_3 <= "1100";
            ELSE
                signal_integer1 <= conv_integer(unsigned(data)) MOD 10;
                signal_integer2 <= (conv_integer(unsigned(data)) / 10) MOD 10;

                BCD_digit_1 <= conv_std_logic_vector(signal_integer1, N - 1);
                BCD_digit_2 <= conv_std_logic_vector(signal_integer2, N - 1);

                IF (minus_con = '1') THEN -- if MSB is 1
                    BCD_digit_3 <= "1011"; -- minus
                ELSE
                    BCD_digit_3 <= "1010"; -- none
                END IF;
            END IF;
        END IF;
    END PROCESS;
END Structural;
```

14. **BinaryToBCDConverterMultiplier :** เป็นส่วนที่รับค่า Input มาเป็น Clock, minus_con ที่รับค่ามาจาก minus และ data ที่รับค่ามาจาก o ของ SignDetector_Multiplier ซึ่งในวงจรนี้จะเป็นการแปลง data ที่เป็น STD_LOGIC_VECTOR N-bit เป็นจำนวน Integer ในแต่ละหลัก ซึ่งใน Component นี้จะแบ่งเป็น
    a. BCD_digit_1, BCD_digit_2 ที่เป็นการแปลง Integer เป็น STD_LOGIC_VECTOR ขนาด N-1 bit
    b. เมื่อ minus_con = '1' จะทำให้ BCD_digit_3 = "1011" ซึ่งคือเครื่องหมายลบ (Segment g)
    c. เมื่อ minus_con = '0' จะทำให้ BCD_digit_3 = "1010" ซึ่งคือการไม่แสดงผลใด ๆ บน 7-Segment หลักที่ 3




```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.STD_LOGIC_ARITH.ALL;

ENTITY BinaryToBCDConverterMUL IS
    GENERIC (
        N : INTEGER := 5
    );
    PORT (
        clk : IN STD_LOGIC;
        minus_con : IN STD_LOGIC;
        data : IN STD_LOGIC_VECTOR(2 * N - 1 DOWNTO 0);
        rst : IN STD_LOGIC;
        BCD_digit_1 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0)
    );
END BinaryToBCDConverterMUL;

ARCHITECTURE Structural OF BinaryToBCDConverterMUL IS
    SIGNAL signal_integer1 : INTEGER := 0;
    SIGNAL signal_integer2 : INTEGER := 0;
BEGIN
    PROCESS (clk)
    BEGIN
        IF rising_edge(clk) THEN
            IF (unsigned(data) > 99) THEN
                BCD_digit_1 <= "1101";
                BCD_digit_2 <= "1101";
                BCD_digit_3 <= "1100";
            ELSE
                signal_integer1 <= conv_integer(unsigned(data)) MOD 10;
                signal_integer2 <= (conv_integer(unsigned(data)) / 10) MOD 10;

                BCD_digit_1 <= conv_std_logic_vector(signal_integer1, N - 1);
                BCD_digit_2 <= conv_std_logic_vector(signal_integer2, N - 1);

                IF (minus_con = '1') THEN -- if MSB is 1
                    BCD_digit_3 <= "1011"; -- minus
                ELSE
                    BCD_digit_3 <= "1010"; -- none
                END IF;
            END IF;
        END IF;
    END PROCESS;
END Structural;
```

15. **BinaryToBCDConverterDivider :** เป็นส่วนที่รับค่า Input มาเป็น Clock, data_err, data_q, data_r, minus_q และ minus_r โดย data_err รับค่ามาจากขา output ERR ของ BinaryDivider, data_q รับค่ามาจาก Output Quotient ของ BinaryDivider, data_r รับค่ามาจาก Output Remainder ของ BinaryDivider, minus_q รับค่ามาจาก Output MINUS_QUOTIENT ของ BinaryDivider และ minus_r รับค่ามาจาก Output MINUS_REMAINDER ของ BinaryDivider ซึ่งในวงจรนี้จะเป็นการแปลง data ที่เป็น STD_LOGIC_VECTOR N-bit เป็นจำนวน Integer ในแต่ละหลัก ซึ่งใน Component นี้จะแบ่งเป็น
    a. BCD_digit_1, BCD_digit_2 ที่เป็นการแปลงค่าจาก data_q และ data_r เป็น STD_LOGIC_VECTOR ขนาด N-1 bit
    b. BCD_digit_4, BCD_digit_5 ที่เป็นการแปลงค่าจาก minus_q และ minus_r เป็น STD_LOGIC_VECTOR ขนาด N-1 bit
    c. เมื่อ minus_con = '1' จะทำให้ BCD_digit_3 = "1011" ซึ่งคือเครื่องหมายลบ (Segment g)
    d. เมื่อ minus_con = '0' จะทำให้ BCD_digit_3 = "1010" ซึ่งคือการไม่แสดงผลใด ๆ บน 7-Segment หลักที่ 3
    e. เมื่อ minus_con = '1' จะทำให้ BCD_digit_6 = "1011" ซึ่งคือเครื่องหมายลบ (Segment g)
    f. เมื่อ minus_con = '0' จะทำให้ BCD_digit_6 = "1010" ซึ่งคือการไม่แสดงผลใด ๆ บน 7-Segment หลักที่ 6



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.STD_LOGIC_ARITH.ALL;

ENTITY BinaryToBCDConverterDIV IS
    GENERIC (
        N : INTEGER := 5
    );
    PORT (
        clk : IN STD_LOGIC;
        minus_q : IN STD_LOGIC;
        minus_r : IN STD_LOGIC;
        data_err : IN STD_LOGIC;
        data_q : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        data_r : IN STD_LOGIC_VECTOR(2 * N - 1 DOWNTO 0);
        BCD_digit_1 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_2 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_3 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_4 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_5 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0);
        BCD_digit_6 : OUT STD_LOGIC_VECTOR(N - 2 DOWNTO 0)
    );
END BinaryToBCDConverterDIV;

ARCHITECTURE Structural OF BinaryToBCDConverterDIV IS
    -- signal integer quotient
    SIGNAL signal_integer1 : INTEGER := 0;
    SIGNAL signal_integer2 : INTEGER := 0;
    -- signal integer remainder
    SIGNAL signal_integer3 : INTEGER := 0;
    SIGNAL signal_integer4 : INTEGER := 0;
BEGIN
    PROCESS (clk)
    BEGIN
        IF rising_edge(clk) THEN
            --IF (unsigned(data_q) > 11111) THEN -- ERR overflow
              --  BCD_digit_1 <= "1101"; -- R
               -- BCD_digit_2 <= "1101"; -- R
               -- BCD_digit_3 <= "1100"; -- E
               -- BCD_digit_4 <= "1101"; -- R
              --  BCD_digit_5 <= "1101"; -- R
              --  BCD_digit_6 <= "1100"; -- E
            -- ELSIF (unsigned(data_r) > 0001100011) THEN -- ERR overflow
              --  BCD_digit_1 <= "1101"; -- R
              --  BCD_digit_2 <= "1101"; -- R
              --  BCD_digit_3 <= "1100"; -- E
              --  BCD_digit_4 <= "1101"; -- R
              --  BCD_digit_5 <= "1101"; -- R
              --  BCD_digit_6 <= "1100"; -- E
            IF (data_err = '1') THEN -- ERR div by zero
                BCD_digit_1 <= "1101"; -- R
                BCD_digit_2 <= "1101"; -- R
                BCD_digit_3 <= "1100"; -- E
                BCD_digit_4 <= "0000"; -- 0
                BCD_digit_5 <= "0000"; -- 0
                BCD_digit_6 <= "0000"; -- 0
            ELSE -- normal mod and div operations
                signal_integer1 <= conv_integer(unsigned(data_q)) MOD 10;
                signal_integer2 <= (conv_integer(unsigned(data_q)) / 10) MOD 10;
                signal_integer3 <= conv_integer(unsigned(data_r)) MOD 10;
                signal_integer4 <= (conv_integer(unsigned(data_r)) / 10) MOD 10;

                BCD_digit_1 <= conv_std_logic_vector(signal_integer1, N - 1);
                BCD_digit_2 <= conv_std_logic_vector(signal_integer2, N - 1);
                BCD_digit_4 <= conv_std_logic_vector(signal_integer3, N - 1);
                BCD_digit_5 <= conv_std_logic_vector(signal_integer4, N - 1);

                IF (minus_q = '1') and (minus_r = '0') THEN
                    BCD_digit_3 <= "1011";
                          BCD_digit_6 <= "1111";
                ELSIF (minus_q = '0') and (minus_r = '1') THEN
                          BCD_digit_3 <= "1111";
                    BCD_digit_6 <= "1011";
                ELSIF (minus_q = '1') and (minus_r = '1') THEN
                    BCD_digit_3 <= "1011";
                    BCD_digit_6 <= "1011";
                     ELSE
                    BCD_digit_3 <= "1111";
                    BCD_digit_6 <= "1111";
                END IF;
            END IF;
        END IF;
    END PROCESS;
END Structural;

```